打包部署 Spring Boot 项目

Spring Boot 使用了内嵌容器，因此它的部署方式也变得非常简单灵活，一方面可以将 Spring Boot 项目打包成独立的 Jar 或者 War 包来运行，也可以单独打包成 War 包部署到 Tomcat 容器中运行，如果涉及到大规模的部署 Jinkins 就成为最佳选择之一。

Spring Boot 默认集成 Web 容器，启动方式和普通 Java 程序一样，main 函数入口启动，其内置 Tomcat 容器或 Jetty 容器，具体由配置来决定（默认 Tomcat）。

相关配置

- 多环境配置

在这里将介绍一下 Spring Boot 多环境配置文件，在我们开发过程中必定会面临多环境的问题，比如开发环境、测试环境、生产环境，在不同的环境下会有不同的数据库连接池等配置信息。如果都写在一个配置文件中，在不同的环境下启动需要手动修改对应的环境参数，这种方式容易出错且不够优雅。Spring Boot 支持多配置文件的使用，只需要启动时指定对应的配置文件即可。

首先在 pom.xml 中添加相关配置：

```
<profiles>
    <profile>
        <id>dev</id>
        <properties>
            <env>dev</env>
        </properties>
        <activation>
            <activeByDefault>true</activeByDefault>
        </activation>
    </profile>
    <profile>
        <id>test</id>
        <properties>
            <env>test</env>
        </properties>
    </profile>
    <profile>
        <id>pro</id>
        <properties>
            <env>pro</env>
        </properties>
    </profile>
```

```
</profiles>

<build>
    <resources>
        <resource>
            <directory>src/main/resources</directory>
            <filtering>true</filtering>
            <excludes>
                <exclude>application-dev.properties</exclude>
                <exclude>application-pro.properties</exclude>
                <exclude>application-test.properties</exclude>
            </excludes>
        </resource>
        <resource>
            <directory>src/main/resources</directory>
            <filtering>true</filtering>
            <includes>
                <include>application-${env}.properties</include>
            </includes>
        </resource>
    </resources>
    <plugins>
        <plugin>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-maven-plugin</artifactId>
        </plugin>
    </plugins>
</build>
```

在 Spring Boot 中多环境配置文件名需要满足 application-{profile}.properties 的格式，其中 {profile} 对应环境标识。在 resources 目录下创建以下三个文件。

- pplication-dev.properties：开发环境
- application-test.properties：测试环境
- application-prod.properties：生产环境

不同的配置文件对应不同的环境，启动的时候通过参数设置来启用不同的配置，开发过程中在 application.properties 文件中通过 spring.profiles.active 属性来设置，其值对应 {profile} 值，生产中使用 --spring.profiles.active=dev 参数来控制加载某个配置文件。

配置项介绍

对 Server 的几个常用配置做个简单说明：

```
# 项目 contextPath，一般不用配置
```

```
server.servlet.context-path=/myspringboot
# 错误页，指定发生错误时，跳转的 URL
server.error.path=/error
# 服务端口
server.port=6060
# session最大超时时间(分钟)，默认为30
server.session-timeout=60
# 该服务绑定 IP 地址，启动服务器时如本机不是该 IP 地址则抛出异常启动失败，只有特殊需求的情况下才配置
# server.address=192.168.0.6
```

如果使用的是 Tomcat 还可以进行以下的配置：

```
# tomcat 最大线程数，默认为 200
server.tomcat.max-threads=600
# tomcat 的 URI 编码
server.tomcat.uri-encoding=UTF-8
# 存放 Tomcat 的日志、Dump 等文件的临时文件夹，默认为系统的 tmp 文件夹
server.tomcat.basedir=/tmp/log
# 打开 Tomcat 的 Access 日志，并可以设置日志格式
#server.tomcat.access-log-enabled=true
#server.tomcat.access-log-pattern=
# accesslog 目录，默认在 basedir/logs
#server.tomcat.accesslog.directory=
# 日志文件目录
logging.path=/tmp/log
# 日志文件名称，默认为 spring.log
logging.file=myapp.log
```

项目打包

- 打成 jar 包

内嵌容器技术的发展为 Spring Boot 部署打下了坚实的基础，内嵌容器不只在部署阶段发挥着巨大的作用，在开发调试阶段也会带来极大的便利性，对比以往开发 Web 项目时配置 Tomcat 的繁琐，会让大家使用 Spring Boot 内嵌容器时有更深的感触。使用 Spring Boot 开发 Web 项目，不需要关心容器的环境问题，专心写业务代码即可。

内嵌容器对部署带来的改变更多，现在 Maven 、Gradle 已经成了我们日常开发必不可少的构建工具，使用这些工具很容易的将项目打包成 Jar 或者 War 包，在服务器上仅仅需要一条命令即可启动项目，我们以 Maven 为例来进行演示。

Maven 默认会将项目打成 jar 包，也可以显示指出打包方式。

```
<groupId>com.neo</groupId>
```

```
<artifactId>spring-boot-package</artifactId>
<version>1.0.0</version>
<!--指定打包方式-->
<packaging>jar</packaging>
```

Maven 打包会根据 pom 包中的 packaging 配置来决定是生成 Jar 包或者 War 包。

pom.xml 同目录下，执行以下命令：

```
cd 项目跟目录（和 pom.xml 同级）
mvn clean package
## 或者执行下面的命令
## 排除测试代码后进行打包
mvn clean package  -Dmaven.test.skip=true
```

- mvn clean 是清除项目 target 目录下的文件
- mvn package 打包命令，可以和 mvn clean 一起执行

在使用 Maven 进行打包的时候，会自动对 test 包下面的测试用例进行测试，测试用例失败会给出错误提示，并停止打包，可以利用这个功能来做项目的自动化测试，检测基本功能是否正确，也可以选择不执行这些测试用例，启动时添加参数 -Dmaven.test.skip=true。

打包完成后 jar 包会生成到 target 目录下，命名一般是“项目名+版本号.jar”的形式。

启动 jar 包命令：

```
java -jar  target/spring-boot-package-1.0.0.jar
```

这种方式，只要控制台关闭，服务就会停止，生产中我们一般使用后台运行的方式来启动:

```
nohup java -jar spring-boot-package-1.0.0.jar &
```

设置 jvm 参数：

```
java -Xms10m -Xmx80m -jar spring-boot-package-1.0.0.jar &
```

也可以在启动的时候选择读取不同的配置文件：

```
java -jar spring-boot-package-1.0.0.jar --spring.profiles.active=dev
```

部署运维

单个项目

简单粗暴

直接 kill 掉进程再次启动 jar 包：

```
ps -ef|grep java
##拿到对于 Java 程序的 pid
kill -9 pid
## 再次重启
Java -jar  xxxx.jar
```

当然这种方式比较传统和暴力，建议大家使用下面的方式来管理。

脚本执行

如果使用的是 Maven，需要包含以下的配置：

```
<plugin>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-maven-plugin</artifactId>
    <configuration>
        <executable>true</executable>
    </configuration>
</plugin>
```

- 启动方式：

（1）可以直接 ./yourapp.jar 来启动

（2）注册为服务

也可以做一个软链接指向你的 jar 包并加入到 init.d 中，然后用命令来启动。

init.d 例子:

```
ln -s /var/yourapp/yourapp.jar /etc/init.d/yourapp
chmod +x /etc/init.d/yourapp
```

这样就可以使用 stop 或者是 restart 命令去管理你的应用了。

```
/etc/init.d/yourapp start|stop|restart
或者

service yourapp start|stop|restart
```

批量部署

Jenkins 是目前持续构建领域使用最广泛的工具之一，它是一个独立的开源自动化服务器，可用于自动化各种任务，如构建、测试和部署软件。Jenkins 可以通过本机系统包以 Docker 的方式部署项目，甚至可以通过安装 Java Runtime Environment 的任何机器独立运行。

Jenkins 可以很好的支持各种语言（如 Java、C#、PHP 等）项目构建，也完全兼容 ant、maven、gradle 等多种第三方构建工具，同时跟 svn、git 能无缝集成，也支持直接与知名源代码托管网站，比如 GitHub、Bitbucket 直接集成。

说直白一点 Jenkins 就是专门来负责如何将代码变成可执行的程序包，将它部署到目标服务器中，并对其运营状态（日志）进行监控的软件。自动化、性能、打包、部署、发布、发布结果自动化验证、接口测试、单元测试等关于我们打包测试部署的方方面面 Jenkins 都可以很友好的支持。

Jenkins 搭建部署分为四个步骤：

1. Jenkins 安装
2. 插件安装和配置
3. Push SSH
4. 部署项目

> http://www.ityouknow.com/springboot/2017/11/11/springboot-jenkins.html

在这里主要看一下使用 Jenkins 部署 Spring Boot 的脚本。

```
DATE=$(date +%Y%m%d)
export JAVA_HOME PATH CLASSPATH
JAVA_HOME=/usr/java/jdk1.8.0_131
PATH=$JAVA_HOME/bin:$JAVA_HOME/jre/bin:$PATH
CLASSPATH=.:$JAVA_HOME/lib:$JAVA_HOME/jre/lib:$CLASSPATH
DIR=/root/xxx
JARFILE=xxx-1.0.0-SNAPSHOT.jar

if [ ! -d $DIR/backup ];then
```

```
    mkdir -p $DIR/backup
fi
cd $DIR

ps -ef | grep $JARFILE | grep -v grep | awk '{print $2}' | xargs kill -9
mv $JARFILE backup/$JARFILE$DATE
mv -f /root/Jenkins-in/$JARFILE .

java -jar $JARFILE > out.log &
if [ $? = 0 ];then
        sleep 30
        tail -n 50 out.log
fi
cd backup/
ls -lt|awk 'NR>5{print $NF}'|xargs rm -rf
```

首先导入相关环境变量，然后根据日期对项目进行备份，kill 掉正在执行的 Spring Boot 项目，将推送过来的项目包放到对应目录下，使用 java -jar 命令来启动应用，最后将启动日志打印出来。

使用 Jenkin 之后，部署项目的步骤如下：

- push 代码到 Github（或者 SVN） 触发 WebHook
- Jenkins 从仓库拉去代码
- Maven 构建项目、单元测试
- 备份项目，停止正在运行的项目
- 启动应用
- 查看启动日志

查看应用运行配置
可以根据 Java 自带的 jinfo 命令：

```
jinfo -flags pid
```

来查看 jar 启动后使用的是什么 gc、新生代、老年代分批的内存都是多少，示例如下：

```
Attaching to process ID 5589, please wait...
Debugger attached successfully.
Server compiler detected.
JVM version is 25.131-b11
Non-default VM flags: -XX:CICompilerCount=2 -XX:InitialHeapSize=16777216 -XX:MaxHea
pSize=262144000
 -XX:MaxNewSize=87359488 -XX:MinHeapDeltaBytes=196608 -XX:NewSize=5570560 -XX:OldSi
```

```
ze=11206656
 -XX:+UseCompressedClassPointers -XX:+UseCompressedOops -XX:+UseParallelGC
```

- -XX:CICompilerCount：最大的并行编译数
- -XX:InitialHeapSize 和 -XX:MaxHeapSize：指定 JVM 的初始和最大堆内存大小
- -XX:MaxNewSize：JVM 堆区域新生代内存的最大可分配大小
- XX:+UseCompressedClassPointers: 压缩类指针
- -XX:+UseCompressedOops: 压缩对象指针
- -XX:+UseParallelGC：垃圾回收使用 Parallel 收集器